MATH 573 Reliable Mathematical Software

Sou-Cheng T. Choi & Fred J. Hickernell

Searle 218 @ UC, (872) 588-2118, schoi32@iit.edu, www.iit.edu/~schoi32
E1-208 @ IIT, (312) 567-8983, hickernell@iit.edu, www.iit.edu/~hickernell
Wednesdays, 3:15 – 4:30 pm, in E1-026, beginning September 18

Office hours by appointment; please email us!
Course Information and Documents on Blackboard through MyIIT.

Revised Wednesday 6th November, 2013
Objectives of MATH 573 Reliable Mathematical Software

Students will learn

1. What qualities good mathematical software must have to be useful for a general audience in the medium to long term,
2. What software engineering practices are needed to produce quality mathematical software, and
3. To spot flaws in software written by others.

Students will produce

4. A small software package, or part of a larger package, that follows good mathematical software development practice.
Our Philosophy of This Course

▶ This is a **demanding** graduate course. The workload will not be heavy, but you will need to learn things on your own without a good textbook.

▶ We have a **diverse** student body in terms of major and experience. Some parts of the course will be review for some but new for others.

▶ This is an **experimental** course — not yet polished.

▶ This is a **seminar** course. We expect class participation.

▶ It is assumed that students are familiar with **numerical computations** using software like MATLAB, which is covered in MATH 350.
## Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/18/13</td>
<td>Course introduction, Rationale for reliable software, Software repositories</td>
<td>FJH</td>
</tr>
<tr>
<td>9/25/13</td>
<td>Coding efficiency and robustness</td>
<td>SCTC</td>
</tr>
<tr>
<td></td>
<td>Assignment 1 due</td>
<td></td>
</tr>
<tr>
<td>10/02/13</td>
<td>User interface</td>
<td>FJH</td>
</tr>
<tr>
<td></td>
<td>Assignment 2 due</td>
<td></td>
</tr>
<tr>
<td>10/09/13</td>
<td>Documentation</td>
<td>SCTC</td>
</tr>
<tr>
<td></td>
<td>Assignment 3 due</td>
<td></td>
</tr>
<tr>
<td>10/16/13</td>
<td>Theoretical introduction to GAIL</td>
<td>FJH</td>
</tr>
<tr>
<td></td>
<td>Assignment 4 due</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Project description due</td>
<td></td>
</tr>
<tr>
<td>10/23/13</td>
<td>Testing</td>
<td>SCTC</td>
</tr>
<tr>
<td>10/30/13</td>
<td>Software distribution and installation</td>
<td>FJH</td>
</tr>
<tr>
<td></td>
<td>Assignment 5 due</td>
<td></td>
</tr>
<tr>
<td>11/06/13</td>
<td>Theoretical introduction to MINRES-QLP</td>
<td>SCTC</td>
</tr>
</tbody>
</table>
Schedule

11/13/13  Project presentations by Lan Jiang, Xin Tong, Xuan Zhou, & Jagadees
11/20/13  Project presentations by Yuhan Ding, Yizhi Zhang, & Tiago Silva
Important Legal Matters

While enrolled in MATH 573 Reliable Mathematical Software, you may interact with software packages under development by the instructors and their students. These software packages are being developed for academic, non-commercial use. As part of your coursework, you may be asked to suggest modifications or improvements to these packages, and the authors of the software may wish to incorporate your suggestions. If your contributions are incorporated, you will receive appropriate recognition. If you do not wish to grant to the authors a non-exclusive, perpetual, royalty-free license to your contributions, then you must submit a written notice to that effect to the instructors by September 25, 2013. Otherwise, you shall be presumed to have granted the authors the foregoing described license. The decision to grant or not grant such license will not affect your course grade.
Resource Materials

- *The Elements of MATLAB Style* by Richard K. Johnson [Joh10] QA76.73.M296 J64 2011eb, is an e-book at the Galvin Library; it explains how to write good MATLAB code.

- *Writing Scientific Software: A Guide to Good Style* by Suely Oliveira and David Stewart [OS06] is also an e-book at the Galvin Library; it explains about efficient code and the challenges of large-scale problems.

- We will use software repositories and hosting sites such as GitHub and BitBucket. A good Git and Mercurial client software is SourceTree.
MATLAB is a great software for rapid coding and testing of algorithms. The examples and sample programs presented in class are written in MATLAB.

- MATLAB is available in the IIT computer labs
- Students who wish to have MATLAB on their own personal computers can purchase [MATLAB & Simulink Student Version](https://www.mathworks.com/products/matlab.html) for around $99 from [Mathworks](https://www.mathworks.com/).
Help Learning MATLAB

If you are not familiar with MATLAB, you may wish to look at these resources:

- Interactive tutorials for MATLAB, Simulink, Control Systems, & Signal Processing: Mathworks Tutorials
- Doug’s MATLAB Video Tutorials: Doug’s
- Curriculum examples by subject, department, and links to textbooks, downloadable code, videos, and more: Additional Resources; these include the free downloadable texts by Cleve Moler, the creator of MATLAB
- Matlab tutorials developed at Clarkson University University of Florida
- Recorded webinars on a variety of topics at Recorded Webinars
  - Introduction to MATLAB
  - MATLAB for Excel Users
  - Programming with MATLAB
  - Symbolic Computing Tools for Academia (Upcoming on Oct 3!)
Assessments

Final grades will be Satisfactory/Unsatisfactory.

Exercises (≈ one per week) 35%
Project 65%

Before the end of the semester, please fill out the Class Feedback Form the Chicago way (early and often).
Assignment 1 (due 9/25/2013)

- Set up an account on GitHub. Email Lan Jiang (ljiang14@iit.edu) and ask her to add you as a contributor to the GAIL_Dev repository. Download the Git client SourceTree (or any other client that you prefer) and clone a copy of the repository.

- Write a short MATLAB .m file, <your name>Assgn1.m where you include the following:
  - Your name, email, and a paragraph of your aspirations for the course (in comments),
  - A successful call of funappx_g.m, integral_g.m, or meanMC_g.m,
  - A copy of your output (in comments),
  - At least one item in the GAIL library — bug, missing feature, unclear documentation, misspelling etc. — that should be corrected or improved. Again put these in comments.

- Upload your .m file into the folder MATH573/Assignment1.
Assignment 2 (due 10/02/2013)

- Write a short MATLAB .m file, `<your name>Assgn2.m` where you include the following:
  1. Your name and email (in comments)
  2. Your operating system and MATLAB version (in comments)
  3. A successful call of `minresqlp.m`, `csminresqlp.m`, `ssminresqlp.m`, or `shminresqlp.m` in MATH573/MINRES-QLP Pack
  4. A copy of your output (in comments)
  5. Apply at least one MATLAB tool mentioned in this lecture to at least an algorithm in the MINRES-QLP Pack and put the results in comments. Based on the analysis results from the tool(s), make at least one suggestion — bug, missing feature, unclear documentation, misspelling etc. — for correcting or enhancing the code. Put them in comments.

- Upload your .m file into the folder MATH573/Assignment2.
Assignment 3 (due 10/09/2013)

▶ Copy the qeq.m file to become <your name>qeq.m. Include our name and email (in comments).
▶ Improve this function to include input parsing and validation.
   ▶ Allow the coefficients of the quadratic polynomial to be input as qeq(a, b, c) or qeq([a b c]) or qeq(coef), where coef.a, coef.b, and coef.c are the coefficients.
   ▶ Check that the coefficients are numeric and not strings or something else.
   ▶ Provide a warning or an error if the polynomial is linear or constant.
▶ Run your program for one example of each style of input, plus one example that produces a warning or error.
▶ Upload your .m file into the folder MATH573/Assignment3.
Assignment 4 and Project Description (due 10/16/2013)

- Add help to `<your name>qeq.m` from Assignment 3. Include your name and email (in comments).
- The documentation should contain the following elements:
  - One-line summary
  - Syntax of interface(s)
  - Example(s)
  - See also
  - Reference
- Retrieve your help at command line prompt using `help <your name>qeq` to check the content. Use “Wrap Comments” to format the content if necessary.
- Upload your M-file into the folder MATH573/Assignment4.
Assignment 5 (due 10/30/2013)

- In this assignment, you may choose to work with one of the following:
  - `<your name>qeq.m` from Assignment 4
  - GAIL algorithm, duplicated as `<your name><algo>_g.m`
  - MINRES-QLP Pack, duplicated as `<your name><algo>qlp.m`

- Add a doctest to each example in your chosen algorithm M-file. Include your name and email (in comments). Make sure every doctest passes successfully. Copy and paste your test execution results in comments.

- Add a unit test to a new file named `ut_<your name><alog>.m`. Make sure it passes successfully. Copy and paste your test execution results in comments.

- Upload your M-file into the folder `MATH573/Assignment5`. 
Assignment 6 (due 11/06/2013)

- Take your `<your name>`qeq.m file and place it in a folder with i) a workout file, ii) a README.txt file describing `<your name>`qeq.m and including a reference for those citing it, and iii) a Contents.m file.
- Zip the folder and place it on an ftp site where it can be downloaded.
- In the folder MATH573/Assignment6 create a file

  `<your name>DownloadInstall.m`

that downloads your folder and adds your folder’s path to MATLAB’s directory so that if your workout file is run, it will execute without having to add or change the path.
Project

You are required to do a project. Each student’s project must be different. Specifically, you must

- Submit a half-page outline and/or description of the project proposal and scope to instructors by October 16, 2013.
- Upload your text file with the name `<your name>pd.txt` into the folder MATH573/ProjectDescription
- Implement a piece of software that is self-contained or implement a new algorithm or set of features for a larger library. Your software should exhibit the principles discussed in this class.
- Present your work during one of the last two classes. Your presentation should include citations for key related work.
- Complete the final software by December 2, 2013.
## Last Two Weeks

<table>
<thead>
<tr>
<th>Presenter</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadees</td>
<td>RBF-FD: Radial basis functions with finite difference method</td>
</tr>
<tr>
<td>Lan</td>
<td>CubMC: multidimensional integral by Monte Carlo</td>
</tr>
<tr>
<td>Tiago</td>
<td>Rare even probability estimation using QMC and IPaS</td>
</tr>
<tr>
<td>Xin</td>
<td>Optimization with guaranteed error tolerance</td>
</tr>
<tr>
<td>Xuan</td>
<td>Maintainance of the GAIL Repository</td>
</tr>
<tr>
<td>Yizhi</td>
<td>Extending ( \text{integral}_g ) from ([0, 1]) to ([a, b])</td>
</tr>
<tr>
<td>Yuhan</td>
<td>Extending ( \text{funappx}_g ) from ([0, 1]) to ([a, b])</td>
</tr>
</tbody>
</table>
Fred Hickernell’s Personal Introduction

Some things about me that might interest you:

▶ BA in mathematics and physics from Pomona College, PhD in applied mathematics from MIT
▶ Faculty member at University of Southern California (4 years), Hong Kong Baptist University (HKBU) (19 years), and IIT (8 years)
▶ Department head/chair at HKBU (13 years) and IIT (8 years)
▶ Research in nonlinear waves and stability of fluids (many years ago), Monte Carlo methods, experimental design, machine learning, automatic algorithms
▶ Married to Elaine for 34 years, one son and one daughter
▶ Chinese name is 葉扶德
▶ Conversant in Cantonese (love to practice), but only know 2ε Mandarin
▶ Christian
Topics that Fred Likes to Discuss

If you share an interest in these topics, let’s find a time to talk outside of class.

- My areas of mathematical research interest.
- How to succeed in academics or industry
- How I got my Chinese name, good Chinese restaurants in Chicago, anything Chinese
- Music, tennis
- What is our purpose in life? How to know God?
Sou-Cheng’s Goals for MATH 573 RMS

▶ To co-instruct and develop the course materials in solidarity with Fred
▶ To prepare applied and computational math students to be better RAs and more attractive candidates for positions in software industry
▶ To challenge myself
Staunch Scientific Software (SSS)

In the course of our studies, research or work, we may develop some new numerical algorithm, or we may be called upon to implement some numerical algorithm. Just like good ideas needs to be published as research papers, good algorithms need to be published. We should develop

- **Staunch** — watertight, sound, strongly built, substantial, steadfast in principle
- **Scientific Software** — software for solving mathematical, scientific, engineering, and business problems

Staunch Scientific Software is (ideally)

- Accessible
- Justified
- Efficient
- Robust
Staunch Scientific Software Is Accessible

- Algorithms are implemented **transparently** so that others can verify that they have been implemented correctly and modify as needed.
- Code is **easy to follow** with inline documentation to explain each step.
- The **user interface** is convenient and documented so that users know how to use the software and combine it with other software.
- **Parameters** that determine accuracy, robustness, or performance are clearly identified for the user.
- **Case studies** illustrate the performance of the software.
Algorithms have theoretical guarantees of success, i.e., sufficient conditions on the inputs under which the algorithm returns the answer within the specified error tolerance.

Where possible, algorithms have sufficient conditions for failure.

Algorithms have theoretical upper and lower bounds on their computational cost.
Staunch Scientific Software Is Efficient

- Algorithms have an optimal order of computational cost for the problem to be solved and the class of inputs considered.
- Implementation of the algorithm uses the least possible amount of time and memory.
- The implementation can be tuned to match the system architecture on which the software is being run.
Staunchn Scientific Software Is Robust

- The software gives **reproducible** results.
- The software is **tested** under a variety of conditions to ensure that it works as advertised.
- The software provides reasonable answers for a **wide variety of possible inputs**.
- The software catches unacceptable inputs and provides instructive **warning or error messages**.
Why Do We Want to Store Our Software in a Repository?

- Repositories contain an up-to-date copy of our software that does not disappear when your computer crashes.
- Repositories, along with their associated version control software, facilitate collaboration among several authors or developers.
- Repositories make our work available to those we want to be able to get it, such as collaborators (read/write) and the public (read only).

There are several software repository protocols, e.g., CVS, SVN, Git, and Mercurial.
The Structure of a Repository with Version Control

- Copies of the repository can be reside on many different computers, but there is typically one easily accessible host that users can connect to.
- The version control software keeps track of changes made by various users. This allows one to know whether his or her copy is up-to-date.
- Independent copies of the software that have not been merged are branches. Frequent committing, pushing and pulling keeps merges the branches.
- If a new program does not work well, the version control software allows one to revert to the old one.

We will be working with the GAIL_Dev repository residing on github.com. GAIL stands for Guaranteed Automatic Integration Library.
Objectives of Lecture 2

We will

1. review important ideas from Lecture 1 and MATH 350
2. discuss efficient coding
3. consider secure coding
Updates in Lecture 1 slides

1. My UC contact information has been changed to IIT’s.
2. The slide titled “Schedule (Subject to Change)” has been changed.
3. A broken webinar link has been changed to an upcoming MATLAB webinar (on Oct 3)
4. In the subsection “Assessments” under the section “Introduction”, added Assignment 2 and Sou-Cheng’s personal info and goals
Recap of Lecture 1: RRR via SSS

- Defining principle of Reproducible research (RR) is “When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures.” [Cla, BD95]

- RR can be made substantially more reliable—hence reliable reproducible research (RRR)—by staunch scientific software (SSS), which we define as a conceptual framework that encompasses a collection of software development methods for promoting the reliability of reproducing provably correct results in computational sciences.” [Cho13c]

- Defining properties of SSS:
  - Accessible
  - Justified
  - Efficient
  - Robust or Durable

- A repository can enhance accessibility.
Recap of Lecture 1: text vs. binary files in repository

On 9/19/13 11:57 AM, Fred Hickernell wrote:
Thanks to Jagadeeswaran for pointing out that Git handles binary files poorly. In our workflow we need to store binary files in a common location with our text files. These binaries may be .pdf files of papers, .eps files used to create our papers and presentations, or .mat files of output. So we need to find a solution.

After spending some time on the internet, I have come across `git-annex`, `git-fat`, `git-bigiles`. Also, there is some discussion about the `git filter-branch` for really removing a file from the repository.

We need someone to work out a pleasant solution for us that is easy to continue using. I am looking for volunteers and am willing to compensate a successful solution with credit towards one or more assignments, depending on the difficulty.
Team Working with a Repository: Pull before Push

1. Pull
2. Push
3. Edit
4. Merge

schoi32@iit.edu, hickernell@iit.edu
Recap of MATH 350: Well-Posed Problem

- Problem \( f : x \mapsto y \)
- Well-posed problem: one whose solution exists and is unique, and depends continuously on data.
- Ill-posed problem: one that is not well-posed
- Algorithm \( \hat{f} : \hat{x} \mapsto \hat{y} \)
Recap of MATH 350: Backward-Stable Algorithm

Assume input and output spaces are normed spaces.

- **Forward error**: \( \| \Delta y \| = \| \hat{y} - y \| \).
- **Backward error**: \( \| \Delta x \| = \| \hat{x} - x \| \).
- **Forward error analysis (FEA)**: natural but usually hard as \( y \) usually not known
- **Backward error analysis**: considers \( \hat{y} \) as the exact solution to a nearby problem with perturbed input \( \hat{x} \). Usually easier than FEA, but not necessarily easy.
- **Backward-stable algorithm** if backward error is “small”, i.e., \( \| \Delta x \| = O(\varepsilon \| x \|) \), \( \varepsilon \) is machine precision
Recap of MATH 350: Well-Conditioned Problem

- Condition number of a problem (independent of an algorithm):
  \[ C_f(x) = \frac{\|\Delta y\|}{\|\Delta x\|}. \]
  If \( f \) differentiable, \( C_f(x) = \|J_f(x)\| \)

- Well-conditioned problem: a well-posed problem with small condition number

- \( f(x) = \tan(x) \), \( C_f(x) = |\sec^2(x)| \)
  - \( C_f(\pi/4) \approx 1.6 \), well-posed and well-conditioned problem
  - \( C_f(1.5708) \approx 4.3 \times 10^5 \), well-posed but ill-conditioned problem
  - \( f(\pi/2) \) undefined, ill-posed problem

- Forward error = Condition number \( \times \) Backward error

<table>
<thead>
<tr>
<th>Well-conditioned problem</th>
<th>Ill-conditioned problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward stable algorithm (small BE)</td>
<td>Accurate solution (small FE)</td>
</tr>
<tr>
<td>Unstable algorithm</td>
<td>Inaccurate solution (large FE)</td>
</tr>
</tbody>
</table>
Measures of Efficiency: Complexity Analysis

- As input size $n$ increases towards $\infty$, how much time, memory, FLOPs, or number of function evaluations does an algorithm consume?

- Scenarios: worst case, average case, best case

- Asymptotic:
  - $f(x) = O(g(x))$ if $\exists M > 0 \, \exists x_0 \, \forall x > x_0 \, \text{s.t.} \, |f(x)| \leq M|g(x)|$
  - $f(x) = \Omega(g(x))$ if $\exists M > 0 \, \exists x_0 \, \forall x > x_0 \, \text{s.t.} \, |f(x)| \geq M|g(x)|$

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>time ($\uparrow$)</th>
<th>memory</th>
<th>FLOPs</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extracting first element of an $n$-array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finding minimum in a binary search tree</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Printing all elements in an array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick sort (average, worst)</td>
<td>$O(n \log n)$ , $O(n^2)$</td>
<td>$O(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adding two dense square matrices</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td></td>
</tr>
<tr>
<td>Solving square $Ax = b$ with direct method</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(n^3)$</td>
<td></td>
</tr>
<tr>
<td>Composite trapezoidal rule</td>
<td></td>
<td></td>
<td></td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Traveling salesman</td>
<td>$O(n!)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Measures of Efficiency: $P = NP$?

Million-dollar problem: $P = NP$?

- $P$: set of problems with algorithms that can solve for an answer in polynomial time
- $NP$: set of problems with algorithms that can verify an answer in polynomial time
- $NP$-hard ($NPH$): set of problems with no known algorithms that can verify an answer in polynomial time
- $NP$-complete ($NPC$): intersection of $NP$ and $NPH$

Known: $P \subseteq NP$

If $P = NP$, then $P = NP = NPC$, which implies only two sets: $P$ or $NPH$. 
5-min break
Measures of Efficiency in Practice

- Complexity analysis is asymptotic. Polynomial time algorithms can be inefficient with large coefficient or degree
- **Execution time:** `tic, toc; cputime; profile viewer`
- **Memory usage:** `profile viewer`
- **Number of lines:** Moler’s one-liners
  - **Function definition:** `fibfun3 = @(x) (x + x.^2)/(1 - x - x.^2);`
  - **Trapezoidal rule:** `T = sum(diff(x).*y(1:end-1)+y(2:end))/2`
- **Reuse** of established functions, code, and libraries
Fast, Faster, Fastest

Step vector generation

Which one is the fastest? Which one is the slowest?
Which one is the most readable? Which one is the least readable?

1. 0:0.001:1
2. (0:100)/1000
3. linspace(0,1,1001)

How can we speed up further?
Speeding up

- Language syntax and techniques, e.g., memory preallocation
- IDE
- Code Analyzer
- Profiler
- Coder and mex
- Parallel Matlab and GPU
Preallocation of memory

Source: Techniques for Improving Performance

```matlab
function x = fibonacci(n)
x(1)=0;
x(2)=1;
for i = 1:n
    x(i+2) = x(i+1)+x(i); % memory increasing with i
end

function x = fibonacci2(n)
x=zeros(n,1); % preallocate memory
x(1)=0;
x(2)=1;
for i = 1:n
    x(i+2) = x(i+1)+x(i);
end

clear all; tic, x = fibonacci(100000); toc
clear all; tic, x = fibonacci2(100000); toc
```

>> fibonacci.time
Elapsed time is 0.020113 seconds.
Elapsed time is 0.002493 seconds.
MATLAB IDE and Code Analyzer

Heed warnings and errors:

```
function x = fibonacci(n)
    x(1)=0;
    x(2)=1;
    for i = 1:n
        x(i+2) = x(i+1)+x(i);
    end
```

Line 5: The variable 'x' appears to change size on every loop iteration. Consider preallocating for speed.
MATLAB Profiler

Profile Summary

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot (dark band = self time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode23</td>
<td>1</td>
<td>0.036 s</td>
<td>0.010 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc</td>
<td>4</td>
<td>0.032 s</td>
<td>0.003 s</td>
<td></td>
</tr>
<tr>
<td>funfunprivateobjectsarguments</td>
<td>1</td>
<td>0.016 s</td>
<td>0.013 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;getStartObject1</td>
<td>2</td>
<td>0.015 s</td>
<td>0.002 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;getStartObject1</td>
<td>38</td>
<td>0.013 s</td>
<td>0.000 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;setShortValueObject1</td>
<td>1</td>
<td>0.012 s</td>
<td>0.002 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;num2complex</td>
<td>54</td>
<td>0.010 s</td>
<td>0.003 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;setShortValueObject1</td>
<td>19</td>
<td>0.010 s</td>
<td>0.005 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;getStartObjectN</td>
<td>38</td>
<td>0.008 s</td>
<td>0.002 s</td>
<td></td>
</tr>
<tr>
<td>odeget</td>
<td>11</td>
<td>0.005 s</td>
<td>0.005 s</td>
<td></td>
</tr>
<tr>
<td>funfunprivateobjectsfind</td>
<td>1</td>
<td>0.000 s</td>
<td>0.000 s</td>
<td></td>
</tr>
<tr>
<td>workspacefunc&gt;getAbstractValueSummary</td>
<td>21</td>
<td>0.003 s</td>
<td>0.000 s</td>
<td></td>
</tr>
</tbody>
</table>

Profiling for Improving Performance

Undocumented profiler options
Disasters Due to Computer Errors

- TIME: Top 10 End-of-the-World Prophecies—#7: Y2K
- Amazon’s $23,698,655.93 book about flies
- Nasdaq crash triggers fear of data meltdown
Runtime errors

- Wrong assumptions of syntax feature or MATLAB behavior, e.g., Facts: MATLAB `switch` does not “falls through”; MATLAB evaluates $0^{\infty}$ to 0, $\infty^0$ to 1, intmax − inf to intmin, intmax + inf to intmax

- Infinite loop

  ```matlab
  function x = infinite(n)
  x=zeros(n,1); % preallocate memory
  x(1)=0;
  x(2)=1;
  i = 2;
  while i > 0
      x(i+2) = x(i+1)+x(i); % memory increasing with i
      i=i+1;
  end
  ```

  Code analyzer gave green light and failed to warn developer.

- Out of Memory: x=zeros(10^10,1);

- Array out of bounds: x = 1:4; x(5)
Integer/Floating-Point Number Overflow/Underflow

Unsigned and signed 16-bit `short` or `int16_t` integers in C:

![Image credit](scho32@iit.edu, hickernell@iit.edu)
Integer/ Floating-Point Number Overflow/ Underflow (cont’d)

\[
\begin{align*}
x &= \text{intmax} \ % \ 2147483647 \\
x &= x + 1 \ % \ \text{intmax} \\
x &= \text{intmax} \\
x &= x - 1 \ % \ 2147483646 \\
\end{align*}
\]

\[
\begin{align*}
x &= \text{intmin} \ % \ -2147483648 \\
x &= x + 1 \ % \ -2147483647 \\
x &= \text{intmin} \\
x &= x - 1 \ % \ \text{intmin} \\
\end{align*}
\]

\[
\begin{align*}
x &= \text{realmax} \ % \ 1.797693134862316e+308 \\
x &= x + 1 \ % \ \text{realmax} \\
x &= \text{realmax} \\
x &= x - 1 \ % \ \text{realmax} \\
\end{align*}
\]

\[
\begin{align*}
x &= \text{realmin} \ % \ 2.225073858507201e-308 \\
x &= x + 1 \ % \ 1 \\
x &= \text{realmin}; \\
x &= x - 1 \ % \ -1 \\
\end{align*}
\]
Side Effects or Unreadability of Efficient Code

- $n=2$
- $m=3$
- $p=m==n$ \( \text{is it } (p=m)==n \text{ or } p=(m==n)? \)

- Global variables

- In C/C++: \( x[i++] \)
Strategies for Handling Run-Time Problems

- `assert`
- Heed runtime warnings and errors
- Graceful error handling
- Input parsing and checking
- Debugger
- Polyspace Code Prover (R2013b) for C/C++ code
<table>
<thead>
<tr>
<th>MATLAB Tools</th>
<th>Justified</th>
<th>Efficient</th>
<th>Durable/Robust</th>
<th>Accessible/Intelligible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our knowledge</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Repository</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Editor</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Code Analyzer</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Profiler</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Debugger</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Prover</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Example of Running Profiler (Before)

clear all; tic, for i=1:100, [fax, out_param] = funappxtau.g(@(x) x.^2); end; toc

- Take \( n - 1 \) out of \( \max \)
- Take \( (n - 1)^2 \) out of \( \max \)
- Recall \( x = 0:1/(n-1):1; \) is faster
Example of Running Profiler (After)

```matlab
funappx_g (100 calls, 0.121 sec)
function in file /Users/terrya/GAIL_Dev/GAIL_Matlab/Algorithms/univariate_appx/funappx_g.m
Copy to new window for comparing multiple runs
```

Speed up: 18%

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>197</td>
<td><code>y = f(x);</code></td>
<td>300</td>
<td>0.030 s</td>
<td>25.0%</td>
<td></td>
</tr>
<tr>
<td>254</td>
<td><code>fappx = @(x) interp1(y1,y2,x,...)</code></td>
<td>100</td>
<td>0.023 s</td>
<td>18.8%</td>
<td></td>
</tr>
<tr>
<td>253</td>
<td><code>y1 = f(x1);</code></td>
<td>100</td>
<td>0.015 s</td>
<td>12.5%</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td><code>out_param = funappx_g_param(ou...</code></td>
<td>100</td>
<td>0.015 s</td>
<td>12.5%</td>
<td></td>
</tr>
<tr>
<td>237</td>
<td><code>end;</code></td>
<td>200</td>
<td>0.008 s</td>
<td>6.2%</td>
<td></td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td></td>
<td>0.030 s</td>
<td>25.0%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>0.121 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>
Example of Running Debugger

- **Toobar in MATLAB editor:**

- **MATLAB’s Debug Menu:**

- **Command-line commands:** `keyboard`, `dbstop`
Key Steps of Running Debugger

Example: egfun.m

Key Steps in debugging code in a script or function file:

- Set one or more breakpoints
- Run code
- Examine, experiment
  - Step
  - Step in and out of another function
  - Run more code at command line
- return to finish
- Clear all breakpoints

Advanced debugging techniques:

- Advanced: Always stop if errors/warnings
- Conditional stop
A Review of Robust Coding — The Quadratic Equation

What is wrong with this code

\[
\text{quadeq=@(a,b,c) } (-b +[-1 1]' \times \sqrt{b^2-4*a*c}) / (2*a);
\]

for solving the quadratic equation?
A Review of Robust Coding — The Quadratic Equation

What is wrong with this code

```matlab
quadeq=@(a,b,c) (-b +[-1 1]'*sqrt(b^2-4*a*c))/(2*a);
```

for solving the quadratic equation?

- Mathematically correct
- One line
- Efficiently uses vector notation

That’s good.
A Review of Robust Coding — The Quadratic Equation

What is wrong with this code

\[
\text{quadeq=@(a,b,c) (-b + [-1 1]' * \sqrt{b^2 - 4*a*c})/(2*a);}\
\]

for solving the quadratic equation?

- Mathematically correct
- One line
- Efficiently uses vector notation

That’s good, but

- It is susceptible to cancellation, overflow, and underflow error.
- It does not have a convenient user interface.
A Better Quadratic Equation Solver

function x=qeq(a,b,c)
% x=QEQ(a,b,c) finds the roots of the quadratic equation
% a \cdot x^2 + b \cdot x + c = 0

x=[]; %initialize roots
%% scale the inputs
scale=max(abs([a b c]));
al1=a/scale; %scale coefficients to avoid overflow or underflow
bl1=b/scale;
c1=c/scale;
if scale==0, return, end %zero polynomial
%% compute the roots
term=-(bl1 + sign(bl1)*sqrt(bl1^2-4*a1*c1)); %no cancellation error here
if term~=0 % at least one root is nonzero
    x(1)=(2*c1)/term;
    if a1~=0; x(2)=term/(2*a1); end %second root exists
elseif a1~=0
    x=zeros(2,1);
end
x=sort(x);
How Easily Can the User Interact with the Routine?

Convenience for many users requires work by the developer.

- The user should have flexibility in providing inputs. The routine needs to **parse** the input.
- The user might not know all of the inputs that are needed. The routine needs **default values** for most inputs.
- The user might input some wrong values. The routine needs to **validate** the inputs.
- The user needs **warnings** or **error messages** if input parameters are not correct.
Parsing

Recent versions of MATLAB provide input parsing capability. The general steps are the following:

- `varargin` can be used to represent all of the input arguments.
- `p = inputParser;` creates a parsing object `p`.
- You add validation functions, e.g.,
  - `addRequired(p,'N',@isnumeric);` is used for input parameters in a specified order. It adds a required parameter.
  - `addOptional(p,'T',default.T,@isnumeric)` is used for input parameters in a specified order. It checks that the parameter `T` is numeric, and assigns the value `default.T` if not value is given.
  - `addParamValue(p,'disctype',default.disctype,@isstr);` is used for input parameters that are part of a structure or are in arbitrary order, it checks that the parameter `disctype` is a string, and assigns the value `default.disctype` if not value is given.
Parsing cont’d

- \( \text{p.} \text{StructExpand} = \text{true}; \) expands the structure that is input.
- \( \text{p.} \text{KeepUnmatched} = \text{true}; \) keeps fields in the input that are not necessarily checked by the input parser.
- \( \text{parse}(\text{p}, \text{N}, \text{varargin}\{2: \text{end}\}) \) performs the parsing on the variable \( \text{N} \) plus the rest of the optional inputs \( \text{varargin}\{2: \text{end}\} \).
- \( \text{out} \_ \text{sample} = \text{p.} \text{Results}; \) assigns the results to the structure \( \text{out} \_ \text{sample} \).

See `brownmotion.m` for an example.

Note that in these examples, default values are provided for all parameter values.
Validation

Although the input parsing functions do basic validation, e.g., if the input parameter is of the right kind, more specialized parsing is often needed. You can write code that checks whether the input value has the right range of values. E.g.,

```matlab
if ~isposint(out_sample.N) % number of sample paths should be an integer
    warning(['The number of sample paths should be a positive integer, using ' num2str(default.N)])
end
out_sample.N = default.N;
```

schoi32@iit.edu, hickernell@iit.edu MATH 573 Lecture Notes Revised 11/6/2013 64 / 134
Validation cont’d

```matlab
if ~ischar(out_sample.disctype)
    warning('Discretization type should be a character.')
    out_sample.disctype=[];
end
if any(strcmpi(out_sample.disctype,{'timestep','discrete'}))
    out_sample.disctype='timestep';
elseif any(strcmpi(out_sample.disctype,{'BB','bridge'}))
    out_sample.disctype='BB';
elseif any(strcmpi(out_sample.disctype,{'KL','Karhunen'}))
    out_sample.disctype='KL';
else
    warning(['Discretization type not recognized, using ' default.disctype])
    out_sample.disctype=default.disctype;
end
```

Note in these examples warnings are provided if the parameters are not the type expected.
Review for Interface

- **Elements of a function interface:**
  \[
  [\text{out1}, \text{out2}, \ldots] = \text{myfun}(\text{in1}, \text{in2}, \ldots)
  \]

- **Types of function interfaces in MATLAB:**
  - Main function: first function in a function file
  - Local function/subfunction: function other than the first one in a function file. Cannot access variables in parent function
  - Nested function: can access variables in parent function

```
function p = makeParabola(a,b,c)
% Returns a function handle to a parabola
p = @parabola;

function y = parabola(x)
% Evaluates a parabola at x
y = a*x.^2 + b*x + c;
end
end
```

- **Anonymous Functions:** defined using a `function_handle`, e.g.,
  \[ f = @(x) x.^2; \]
More Examples of MATLAB Interfaces

- Few interfaces are as clean and memorable as $x = A \backslash b$

![Image Credit: Cleve’s Corner](Image Credit: Cleve’s Corner)

- Some MATLAB methods have simple interfaces, e.g., `isempty`, `length`
- Others have multiple interfaces due to
  - overloading, e.g., `disp('Hello!')`, `disp([1 2])`
  - optional input arguments, e.g., `max(X)`, `max(X,[],1)`
  - optional output arguments, e.g., `y=max(X)`, `[y,i]=max(X)`
- Some interfaces are rather long and complex, e.g.,
  ```matlab
  [X,FLAG,RELRES,ITER,RESVEC] = ... 
  GMRES(A,B,RESTART,TOL,MAXIT,M1,M2,X0)
  ```
- Some interfaces have variable number of input or output variables:
  ```matlab
  vararginout=foo(varargin)
  ```
Last Word

If we were allowed to communicate only one thing to expert or application users about our implementation, our choice would be its API, which consists of function name, required and optional inputs, outputs, and our parsing schemes.
Getting Help in MATLAB

- To get help at command line prompt:

  ```matlab
  >> help isempty
  ISEMPTY True for empty array.
  ISEMPTY(X) returns 1 if X is an empty array and 0 otherwise. An empty array has no elements, that is prod(size(X))==0.
  
  Overloaded methods:
  varmat/isempty
  oparray/isempty
  ...
  ```

- To get help for overloaded methods:

  ```matlab
  >> help varmat/isempty
  Copyright 2011 by The University of Oxford and The Chebfun Developers. See http://www.maths.ox.ac.uk/chebfun/ for Chebfun information.
  ```

- See the source files:

  ```matlab
  >> dbtype varmat/isempty
  1 function e = isempty(V)
  2 ```
Getting Help in MATLAB (cont’d)

- To get help for a subfunction or a nested function:

  ```matlab
  function [y]=myfun(x)
  % Help for myfun
  y=mysubfun(x);
  
  function [y]=mysubfun(x)
  % Help for mysubfun.
  y = x + 1;
  
  >> help myfun>mysubfun
  Help for mysubfun
  
  - To get directory-level help:

  >> help GAIL_Matlab
Getting Formatted HTML Help in MATLAB

```matlab
doc isempty

isempty
Determine whether array is empty

Syntax

```matlab
TF = isempty(A)
```

Description

```matlab
TF = isempty(A) returns logical 1 (true) if A is an empty array and logical 0 (false) otherwise. An empty array has at least one dimension of size zero, for example, 0-by-0 or 0-by-5.
```

Examples

```matlab
B = rand(2,2,2);
B(:,:, :) = [];
isempty(B)
ans = 1
```

See Also

```matlab
is*
```

MATLAB’s HELP menus: Product Help or Function Browser

lookfor searches first line of help in all M-files
Types of Documentation

For very complex software:

- **User guide**: Getting started, Applications documentation
- **Developer documentation**: Software requirements, Architectural design, Functional documentation, Technical documentation
- **Administrator’s guide**
Elements of MATLAB User Documentation

- Package-level help,
  - README.txt, e.g., GAIL version 1's README.txt
  - license, e.g., GAIL version 1's LICENSE.m
- Directory-level help in Contents.m, e.g., GAIL version 1's Contents.m
  - One-line summary for this folder
  - One-line summary for each function in this folder
  - One-line summary for each subfolder in this folder
- Function-level help after function declaration, e.g., GAIL version 1's integral.m
  - One-line summary
  - Syntax of interface(s)
  - Example(s)
  - Overloaded methods
  - See also
  - Reference
- Inline help for code, e.g., GAIL version 1's integral.m
Creating MATLAB Documentation

- Three types of comments:
  - One-line comment:
    ```matlab
    function myfun(x)
    \{
    \}
    ```
  - Multiline comment:
    ```matlab
    This is a multiline comment.
    \}
    ```
  - One-line title for a section of code:
    ```matlab
    % One-line comment
    ```

- Format long comments using “Wrap Comments”, e.g., blockCom.m

- Beautify code using “Smart Indent” and “…”

- Generating directory-level help file Contents.m
Creating Searchable Customized HTML Documentation

1. Create a subdirectory for HTML documentation, e.g., Documentation
2. Create M-files with comments
   - A package description, e.g., GAIL.m
   - A list of main functions, e.g., funclist.m
   - An M-file for each main function, e.g., help_integral_g.m (Cf. help integral_g.m)
3. Create subdirectory called html
4. Create HTML-files in the subdirectory html for each M-file e.g.,
   publish('GAIL.m', 'html')
5. Create XML-files
   - info.xml: Package metadata
   - html/helptoc.xml: Define navigation menu for all HTML files
6. Build search index and help content by
   builddocsearchdb(strPathToHtmlDirectory)
7. View or search under Product Help
   schoi32@iit.edu, hickernell@iit.edu
Documentation of Partially Automatic Procedures or Routines

- **Examples:**
  - Creating searchable customized HTML documentation
  - Repacking and pruning large binaries in a repository

- **Containers:**
  - Developer guide/Technical documentation
  - Script file
  - Wiki
Preview

Next Wed:

► HW4 due
► Project description due
► Fred on GAIL
The Need for Automatic Algorithms

The user would like to provide the error tolerance $\epsilon$ and a black-box function evaluator or random number generator to the algorithm $\hat{\text{INT}}$ or $\hat{\text{APP}}$ or $\hat{\text{MEAN}}$, and obtain

$$\left| \int_{0}^{1} f(x)dx - \hat{\text{INT}}(f, \epsilon) \right| \leq \epsilon$$

$$\sup_{0 \leq x \leq 1} \left| f(x) - \hat{\text{APP}}(f, \epsilon)(x) \right| \leq \epsilon$$

$$\text{Prob} \left[ \left| \mathbb{E}(Y) - \hat{\text{MEAN}}(Y, \epsilon) \right| \leq \epsilon \right] \geq 99\%$$

guaranteed. How can we make this happen?

GAIL provides $\text{integral}_g.m$, $\text{funappx}_g.m$, and $\text{meanMC}_g.m$. 
Guaranteed Automatic Algorithms Based on Balls

Since we already know that . . .

$$\left| \int_0^1 f(x) \, dx - \text{trap}_n(f) \right| \leq \frac{\|f''\|_1}{8n^2}$$

We choose . . .

$$\hat{\text{INT}}(f, \varepsilon)(f) = \text{trap}_n(f) \quad \text{with}$$

$$n = \left\lceil \sqrt{\frac{\sigma}{8\varepsilon}} \right\rceil \quad \text{provided} \quad \|f''\|_1 \leq \sigma$$

$$\hat{\text{APP}}(f, \varepsilon)(f) = \text{spline}_n(f) \quad \text{with}$$

$$n = \left\lceil \sqrt{\frac{\sigma}{8\varepsilon}} \right\rceil \quad \text{provided}$$

$$\|f''\|_\infty \leq \sigma$$

\[\text{schoi32@iit.edu, hickernell@iit.edu}\]

MATH 573 Lecture Notes

Revised 11/6/2013
Guaranteed Automatic Algorithms Based on Balls

Since we already know that \ldots \quad \textbf{We choose} \ldots

\[ \text{Prob} \left[ \left| \mathbb{E}(Y) - \frac{1}{n} \sum_{i=1}^{n} Y_i \right| \leq \sqrt{\frac{100 \text{var}(Y)}{n}} \right] \geq 99\% \]

\[ \overline{\text{MEAN}}(Y, \varepsilon)(f) = \frac{1}{n} \sum_{i=1}^{n} Y_i \quad \text{with} \]

\[ n = \left\lceil \frac{100 \sigma^2}{\varepsilon^2} \right\rceil \quad \text{provided} \]

\[ \text{var}(Y) \leq \sigma^2 \]

Here we assume an upper bound on the size of $f$ or the variance of $Y$. These are not adaptive.
Adaptive Automatic Algorithms Based on Heuristics or Asymptotics

We assume that ... So we choose ...

\[
\left| \int_0^1 f(x) \, dx - \text{trap}_n(f) \right| \\
\approx \frac{\text{trap}_n(f) - \text{trap}_{n/2}(f)}{3}
\]

\[
\int(f,v)(f) = \text{trap}_n(f) \quad \text{with} \quad \frac{\text{trap}_n(f) - \text{trap}_{n/2}(f)}{3} \leq \varepsilon
\]

\[
\frac{1}{n} \sum_{i=1}^{n} Y_i \approx \mathcal{N}(\mathbb{E}(Y), \text{var}(Y)/n)
\]

\[
\widehat{\text{MEAN}}(Y, \varepsilon)(f) = \frac{1}{n} \sum_{i=1}^{n} Y_i \quad \text{with} \quad \text{var}(Y) \leq \sigma^2
\]

\[
n = \left\lceil \left( \frac{2.58\sigma}{\varepsilon} \right)^2 \right\rceil \quad \text{provided}
\]
Why Algorithms Based on Heuristics and Asymptotics Fail

These functions fool MATLAB’s `quad` algorithm (adaptive Simpson’s rule [GG00]) even with $\varepsilon = 10^{-14}$: $\int_0^1 f_{\text{spiky}}(x) \approx 0.4571$, and `quad` gives 0, $\int_0^1 f_{\text{fluky}}(x) \approx 0.1148$, and `quad` gives 0.1640,
For Guaranteed, Adaptive, Automatic Algorithms Choose Cones, not Balls! [HJLO14, CDH+13]

Problem: How to reliably bound $\|f''\|_1, \|f''\|_\infty, \text{var}(Y)$.

Solution:

- Define a cone of functions where a weaker measure bounds a stronger measure:

  $$\|f''\|_1 \leq \tau \|f' - f(1) + f(0)\|_1, \quad \|f''\|_\infty \leq \tau \|f' - f(1) + f(0)\|_\infty,$$
  $$\mathbb{E}\{[Y - \mathbb{E}(Y)]^4\} \leq \tau [\text{var}(Y)]^2,$$

- Use a data-driven approximation to the weaker measure to reliably bound the stronger measure

- Increase $n$ until the convergence criterion is satisfied.
For Guaranteed, Adaptive, Automatic Integration

- Pick a weaker derivative,

\[ 2 \| f' - f(1) + f(0) \|_1 \leq \| f'' \|_1 \]

always
For Guaranteed, Adaptive, Automatic Integration

- Pick a weaker derivative, and assume that $f$ lies in a cone:

$$2 \| f' - f(1) + f(0) \|_1 \leq \| f'' \|_1 \leq \tau \| f' - f(1) + f(0) \|_1.$$
For Guaranteed, Adaptive, Automatic Integration

- Pick a weaker derivative, and assume that $f$ lies in a cone:

$$2 \| f' - f(1) + f(0) \|_1 \leq \| f'' \|_1 \leq \tau \| f' - f(1) + f(0) \|_1.$$  

- Use data-driven $V_n(f) = \| (\text{linear spline of } f)' - f(1) + f(0) \|_1$, to reliably bound $\| f' - f(1) + f(0) \|_1$, and then $\| f'' \|_1$:  

![Graph showing linear spline and the function $f$.]
For Guaranteed, Adaptive, Automatic Integration

- Pick a weaker derivative, and assume that $f$ lies in a cone:

\[
2 \| f' - f(1) + f(0) \|_1 \leq \| f'' \|_1 \leq \tau \| f' - f(1) + f(0) \|_1.
\]

- Use data-driven $V_n(f) = \| \text{(linear spline of } f)') - f(1) + f(0) \|_1$, to reliably bound $\| f' - f(1) + f(0) \|_1$, and then $\| f'' \|_1$:

\[
0 \leq \| f' - f(1) + f(0) \|_1 - V_n(f) \leq \frac{\| f'' \|_1}{2n} \leq \frac{\tau \| f' - f(1) + f(0) \|_1}{2n}
\]

\[
\tilde{V}_n(f) \leq \| f'' \|_1 \leq \tau \| f' - f(1) + f(0) \|_1 \leq \frac{2\tau V_n(f)}{2 - \tau/n} \leq \frac{\tau \| f'' \|_1}{2 - \tau/n}
\]
For Guaranteed, Adaptive, Automatic Integration

- Pick a weaker derivative, and assume that $f$ lies in a cone:

\[ 2 \| f' - f(1) + f(0) \|_1 \leq \| f'' \|_1 \leq \tau \| f' - f(1) + f(0) \|_1. \]

- Use data-driven $V_n(f) = \| (\text{linear spline of } f)' - f(1) + f(0) \|_1$, to reliably bound $\| f' - f(1) + f(0) \|_1$, and then $\| f'' \|_1$:

\[
0 \leq \| f' - f(1) + f(0) \|_1 - V_n(f) \leq \frac{\| f'' \|_1}{2n} \leq \frac{\tau \| f' - f(1) + f(0) \|_1}{2n}
\]

\[
\tilde{V}_n(f) \leq \| f'' \|_1 \leq \tau \| f' - f(1) + f(0) \|_1 \leq \frac{2\tau V_n(f)}{2 - \tau/n} \leq \frac{\tau \| f'' \|_1}{2 - \tau/n}
\]

- The $n$ needed to meet the desired tolerance is any one satisfying this data-driven criterion

\[
\frac{\tau V_n(f)}{4n(2n - \tau)} \leq \varepsilon.
\]
Hump Function

\[
\int_0^1 f_{\text{hump}}(x) \, dx = 1,
\]

width and center random, 10000 replications,
\[\varepsilon = 10^{-8}, \quad N_{\text{max}} = 10^7\]

<table>
<thead>
<tr>
<th>(\tau)</th>
<th>(\text{Prob}(f \in C_\tau))</th>
<th>(\text{Success})</th>
<th>(\text{Warning})</th>
<th>(\text{Failure})</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0% → 25%</td>
<td>25%</td>
<td>&lt; 1%</td>
<td>75%</td>
</tr>
<tr>
<td>\text{integral}_g_m</td>
<td>100</td>
<td>23% → 58%</td>
<td>56%</td>
<td>2%</td>
</tr>
<tr>
<td>1000</td>
<td>57% → 88%</td>
<td>68%</td>
<td>20%</td>
<td>12%</td>
</tr>
<tr>
<td>\text{quad}</td>
<td></td>
<td></td>
<td>8%</td>
<td>92%</td>
</tr>
<tr>
<td>\text{integral}</td>
<td></td>
<td></td>
<td>19%</td>
<td>81%</td>
</tr>
<tr>
<td>\text{chebfun}</td>
<td></td>
<td></td>
<td>29%</td>
<td>71%</td>
</tr>
</tbody>
</table>
Computational Cost (# of Function Values) for integral $g.m$

$$\sqrt{\frac{(\tau - 2) \|f''\|_1}{32\tau\varepsilon}} - 1 \leq \text{cost of best algorithm for } f \in \mathcal{C} \text{ with known } \|f''\|_1$$

$$\leq \text{cost of trapezoidal rule with known } \|f''\|_1 \approx \sqrt{\frac{\|f''\|_1}{8\varepsilon}} + 1$$

$$\leq \max \left( \sqrt{\frac{\|f''\|_1}{8\varepsilon}}, \frac{\tau + 1}{2} \right) + 1 \leq \text{cost of our algorithm for } f \in \mathcal{C} \text{ with unknown } \|f''\|_1 \leq \sqrt{\frac{\tau \|f''\|_1}{4\varepsilon}} + \tau + 4$$
Some Features of the GAIL Algorithms

- Since initial sample size used by the algorithm is more intuitive than $\tau$, GAIL asks for the initial sample size and infers $\tau$.
- All GAIL algorithms have sample size budgets; \texttt{meanMC.g.m} also has a time budget.
- \texttt{integral.m.g} and \texttt{funappx.g.m} have a data-driven necessary test for $f \in \mathcal{L}$, namely,

$$\tilde{V}_n(f) \leq \frac{2\tau V_n(f)}{2 - \tau/n}$$

where $\tilde{V}_n(f) = \text{Var}((\text{linear spline of } f)'') = \| (\text{linear spline of } f)'' \|_1$. If this test fails, $\tau$ is increased.
Future Features for GAIL

- Lower complexity bounds for mean of a random variable
- Relative error
- $[0,1] \rightarrow [a,b]$ for `integral_g.m` (Yizhi Zhang) and `funappx_g.m` (Yuhan Ding)
- Higher order methods for `integral_g.m` and `funappx_g.m`
- Local adaption for `integral_g.m` and `funappx_g.m`
- `meanMCBinomial_g.m`, `meanMCCV_g.m`
- `cubMC_g.m` (Lan Jiang), `cubMCIS_g.m`
- `meanQMC_g.m` (Tony Jiménez Rugama), `meanMLMC_g.m` (Aleks Borkovskiy)
- `funmin_g.m` (Xin Tong)
- `funappxmulti_g.m` (Xuan Zhou)
- ??? (you)
A Nagging Question

For example, for integral_g.m the convergence criterion is

\[
\frac{\tau V_n(f)}{4n(2n - \tau)} \leq \varepsilon.
\]

But \( \tau V_n(f) \) may be a big overestimate of \( \|f''\|_1 \); \( \tilde{V}_n(f) \) may be adequate. If we replaced this convergence criterion by

\[
\frac{C(n)\tilde{V}_n(f)}{8n^2} \leq \varepsilon, \quad n \geq N_{\text{min}}, \quad C(n) \downarrow 1 \text{ as } n \uparrow \infty
\]

would it still be reasonable?
Solving Large Singular Linear System of Equations

Given \( A = A^T \in \mathbb{R}^{n \times n}, \ b \in \mathbb{R}^n \ (n >> 1) \). Want to solve \( Ax = b \) for \( x \), e.g.,

\[
A = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad b = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
\]

Minimum-length (pseudoinverse) solution is:

\[
x^\dagger = \begin{bmatrix}
1 \\
1 \\
0
\end{bmatrix} \quad \|x^\dagger\|_2 \approx 1.4 \quad r^\dagger = b - Ax^\dagger = \begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix} \quad Ar^\dagger = 0
\]

A least-squares solution is:

\[
x^\# = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix} \quad \|x^\#\|_2 \approx 1.7 \quad r^\# = b - Ax^\# = \begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix} \quad Ar^\# = 0
\]

The proper residual norm to consider is not \( \|r\| \), but \( \|Ar\| \).
MATLAB Iterative Solvers Fail

- Conjugate gradient (CG) does not converge to a solution:
  \[ A = \begin{bmatrix} 1 & 0 & 0; & 0 & 1 & 0; & 0 & 0 & 0 \end{bmatrix}, \ b = \begin{bmatrix} 1; & 1; & 1 \end{bmatrix}, \ x = \text{pcg}(A, b) \]
  \[
  x = \begin{bmatrix} 1.5 \\ 1.5 \\ 1.5 \\ 1.5 \end{bmatrix}, \quad \|Ar\| = 0.7071
  \]

- SYMMLQ returns an exploding solution: \[ x = \text{symmlq}(A, b) \]
  \[
  x = 10^{16} \begin{bmatrix} 0 \\ 0 \\ 2.2063 \end{bmatrix}, \quad \|Ar\| = 1.4142
  \]

- MINRES returns an exploding solution: \[ x = \text{minres}(A, b) \]
  \[
  x = 10^{15} \begin{bmatrix} 0 \\ 0 \\ 5.1913 \end{bmatrix}, \quad \|Ar\| = 0.3328
  \]
MINRES\(^{(2)}\) and MINRES-QLP

[Cho06, CPS11, CS]

- **MINRES\(^{(2)}\)** returns \(x^\#\):
  \[
x = \begin{bmatrix} 
  1 \\
  1 \\
  1
  \end{bmatrix}, \quad \|Ar\| = 3.1402 \times 10^{-16}
  \]

- **MINRES-QLP** returns \(x^\dagger\):
  \[
x = \begin{bmatrix} 
  1 \\
  1 \\
  1.5701 \times 10^{-16}
  \end{bmatrix}, \quad \|Ar\| = 3.1402 \times 10^{-16}
  \]
Small Talk

THIS IS A SINGULAR SYSTEM. WE CANNOT SOLVE IT!

YES, WE CAN!
Lanczos process

\[ v_0 = 0 \quad \beta_1 v_1 = b \quad (\beta_k \text{ normalizes } v_k) \]

Compute vectors \( v_k \) until \( \beta_{k+1} = 0 \):

\[ p_k = Av_k \quad \alpha_k = v_k^T p_k \]
\[ \beta_{k+1} v_{k+1} = p_k - \alpha_k v_k - \beta_k v_{k-1} \]

In matrix form:

\[ AV_k = V_{k+1} T_k \quad V_k = \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix} \]

\[ T_k = \begin{bmatrix} \alpha_1 & \beta_2 \\ \beta_2 & \alpha_2 & \ddots \\ \vdots & \ddots & \ddots & \beta_k \\ \beta_k & \alpha_k & \beta_{k+1} & \end{bmatrix} =: \begin{bmatrix} T_k \\ \beta_{k+1} e_k^T \end{bmatrix} \]

\( k \)th Krylov subspace \( K_k(A, b) = \mathcal{R}(V_k) \)

\[ \text{rank}(T_k) = \begin{cases} k & \text{if } b \in \mathcal{R}(A) \\ k - 1 \text{ or } k & \text{otherwise} \end{cases} \]
### Krylov methods for Symmetric linear systems

<table>
<thead>
<tr>
<th>Method</th>
<th>Subproblem</th>
<th>Factorization</th>
<th>Estimate of $x_k$</th>
</tr>
</thead>
</table>
| CG      | $T_k y_k = \beta_1 e_1$                                                     | Cholesky               | $x_k = V_k y_k$ 
$T_k = L_k D_k L_k^T$ 
$\in K_k(A, b)$                   |
| SYMMLQ  | $y_{k+1} = \arg\min \|y\| \mid \underline{T_k}^T y = \beta_1 e_1$          | LQ                     | $x_k = V_{k+1} y_{k+1}$ 
$\in K_{k+1}(A, b)$                                          |
| MINRES  | $y_k = \arg\min \|T_k y - \beta_1 e_1\|$                                  | QR                     | $x_k = V_k y_k$ 
$Q_k T_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix}$ 
$\in K_k(A, b)$                      |

<table>
<thead>
<tr>
<th>Method</th>
<th>New basis</th>
<th>$z_k$</th>
<th>Estimate of $x_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>$W_k := V_k L_k \underline{^T}$</td>
<td>$L_k D_k z_k = \beta_1 e_1$</td>
<td>$x_k = W_k z_k$</td>
</tr>
<tr>
<td>SYMMLQ</td>
<td>$W_k := V_{k+1} Q_k \begin{bmatrix} I_k \ 0 \end{bmatrix}$</td>
<td>$L_k z_k = \beta_1 e_1$</td>
<td>$x_k = W_k z_k$</td>
</tr>
<tr>
<td>MINRES</td>
<td>$W_k := V_k R_k^{-1}$</td>
<td>$R_k z_k = \beta_1 [I_k \ 0] Q_k e_1$</td>
<td>$x_k = W_k z_k$</td>
</tr>
</tbody>
</table>
MINRES-QLP

In $k$th Lanczos iteration, MINRES-QLP solves subproblem

$$\min \| y_k \| \quad \text{s.t.} \quad y_k \in \arg \min_{y \in \mathbb{R}^k} \| T_k y - \beta_1 e_1 \|_2 \quad x_k = V_k y_k$$

(1) QLP decomposition of tridiagonal $T_k$

$$Q_k T_k P_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix} \quad P_k = \begin{bmatrix} L_k \\ 0 \end{bmatrix}$$

$$Q_k = \cdots Q_{34} \quad Q_{23} \quad Q_{12}$$

$$P_k = P_{12} P_{13} \quad P_{23} P_{24} \quad P_{34} P_{35} \quad \cdots$$

$Q_{i,i+1}$ and $P_{i,i+1}$ are symmetric orthogonal transformations that zero one subdiagonal of $T_k$ and two superdiagonals of $R_k$:

$$Q_{i,i+1} = \begin{bmatrix} I_{i-1} \\ c_i & s_i \\ s_i & -c_i \\ I_{k-i} \end{bmatrix}.$$
MINRES-QLP: Solving $k$th subproblem

(2) Apply QLP to subproblem:

$$y_k = P_k u_k \quad \text{min} \left\| \begin{bmatrix} L_k \\ 0 \end{bmatrix} u_k - \begin{bmatrix} t_k \\ \phi_k \end{bmatrix} \right\| \quad L_k u_k = t_k$$

(a) If $\text{rank}(L_k) = k$, solve bottom 3 eqns of $L_k u_k = t_k$

(b) If $\text{rank}(L_k) = k - 1$, solve bottom 2 eqns of

$$L_{k-1}^{(2)} u_{k-1}^{(2)} = t_{k-1} \quad L_k = \begin{bmatrix} L_{k-1}^{(2)} & 0 \\ 0 & 0 \end{bmatrix} \quad u_k = \begin{bmatrix} u_{k-1}^{(2)} \\ 0 \end{bmatrix}$$

(final iteration)

(3) Change to an orthonormal basis in $K_k(A, b)$, and update solution by orthogonal steps:

$$W_k = V_k P_k = \begin{bmatrix} W_{k-3} & W_k(:, J) \end{bmatrix} \quad J = k - 2 : k$$

$$x_k = V_k y_k = W_k u_k = x_{k-3} + W_k(:, J) u_k(J)$$
MINRES-QLP: Norm estimates

\[ \phi_k := \| r_k \|_2 = \phi_{k-1} s_k \quad \phi_0 = \| b \| \]

\[ \psi_k := \| A r_k \|_2 = \phi_k \left\| \begin{bmatrix} R_{k+1}(k, k+1) \\ R_{k+1}(k + 1, k + 1) \end{bmatrix} \right\| \]

\[ \chi_k := \| x_k \|_2 = \left\| \begin{bmatrix} \chi_{k-3} \\ u_k(J) \end{bmatrix} \right\| \quad \chi_0 = 0 \]

\[ \omega_k := \| A x_k \|_2 = \left\| \begin{bmatrix} \omega_{k-1} \\ t_k(k) \end{bmatrix} \right\| \quad \omega_0 = 0 \]

\[ \| A \|_2 \geq A_2^{(k)} := \max \left\{ A_2^{(k-1)}, \| T_k e_k \|_2 , L_k(i,i) \right\} \quad A_2^{(1)} := \| T_1 e_1 \|_2 \]

\[ \kappa_2(A) \geq \kappa_2(T_k) \approx \frac{\max_i L_k(i,i)}{\min_i L_k(i,i)} \quad \text{(QLP)} \]
MINRES-QLP: Stopping conditions

(1) From Lanczos:

\[ \beta_{k+1} \leq n\|A\|\varepsilon \]
\[ k = \text{MAXIT} \]

(2) Normwise relative backward errors (NRBE):

\[ \frac{\|r_k\|_2}{(\|A\|\|x_k\| + \|b\|)} \leq \varepsilon \]
\[ \frac{\|Ar_k\|_2}{(\|A\|\|r_k\|)} \leq \varepsilon \]

(3) Regularization attempts:

\[ \kappa(A) \geq \text{MAXCOND} \]
\[ \|x_k\|_2 \geq \text{MAXXNORM} \]
Theorems on MINRES and MINRES-QLP

Ax = b or min ∥Ax − b∥  A = A^T, possibly singular

(1) If b ∈ ℜ(A), β_{k+1} = 0, then
   (a) ∥r_k∥ = 0
   (b) x_k from MINRES and MINRES-QLP is the minimum-length solution.

(2) If b ∉ ℜ(A), β_{k+1} = R_{kk} = 0, then
   (a) ∥Ar_k∥ = 0
   (b) x_{k−1} from MINRES is a least-squares solution
   (c) x_k from MINRES-QLP is the minimum-length solution
Laplacian Least-Squares Problem

$$A = \begin{bmatrix} T & T \\ T & T \\ \vdots & \vdots \\ T & T \end{bmatrix} \quad T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix} \quad b = 10z, \quad z(i) \sim U[0, 1]$$

![Graph showing the performance of different solvers for the Laplacian Least-Squares Problem.](image-url)
Summary

- On nonsingular systems, MINRES-QLP potentially returns a more accurate solution than MINRES (because of its orthogonal steps).
- On singular systems, MINRES-QLP returns the minimum-length solution, i.e., simultaneously minimizes the residual and solution norms (conflicting goals)—a feature not present in other iterative methods for singular Hermitian least-squares.
- Moderate additional cost relative to MINRES:
  1 more vector in memory
  4 more saxpy’s \((y \leftarrow \alpha x + y)\) per iteration
  3 more vector scalings \((x \leftarrow \alpha x)\)
- Immediately applicable to complex Hermitian problems \((A = A^*)\).
- 2012 SIAG/Linear Algebra Prize for [CPS11]

conervation law: \(\frac{\$x}{n} = \$x\)
Ongoing Work

- Extended to systems with $A$
  - complex symmetric ($A = A^T \in \mathbb{C}^n$): CS-MINRES-QLP
  - skew-symmetric ($A = -A^T \in \mathbb{R}^n$): SS-MINRES-QLP
  - skew-Hermitian ($A = -A^* \in \mathbb{C}^n$): SH-MINRES-QLP

using different (generalized) Lanczos frameworks [Cho13a]

- Singular nonsymmetric $A$ [Cho13b]: GMRES-QLP, GMRES-URV

- Software (Matlab, Python, Fortran, PETSc): MINRES-QLP Pack [Cho13c]

- Community website: http://code.google.com/p/minres-qlp/

- Upcoming conferences and presentations:
  - Nov 14: UChicago SSC Seminars
  - Nov 17: WSSSPE, Denver
  - Dec 05: Taiwan National U, TIMS Winter School for Scientific Computing
  - Dec 16-18: Hong Kong Polytechnic U, ECM
Review of Software Development Process

- Last week’s theory on GAIL justifies staunch scientific software development
- Yesterday’s MATLAB Technical Seminar explores efficient software development
- Software development cycle:
  
  **In theory . . .**

  ![Diagram of software development cycle in theory](image)

  - Define
  - Design
  - Develop
  - Document
  - Debug
  - Deploy
  - Test
  - Release

  **In practice . . .**

  ![Diagram of software development cycle in practice](image)

  - Define
  - Re/design
  - Develop
  - Document
  - Debug
  - Deploy
  - Test
  - Release
  - Re/define
  - Re/design
Overview of Test-Driven Development (TDD) Process

In essence . . .

In reality . . .

Define
Design
Develop
Document
Debug
Deploy
Release

Test

Write a Failing Test
Make the Test Pass
Refactor

Image Credit: MSDN
Overview of Tests

Why do testing?
- To define and design APIs, and drive software development
- To serve as usage examples and thus a kind of documentation
- To check validity and accuracy (preconditions, post-conditions, intermediate summary statistics)
- To find and reduce errors (a.k.a. “bugs”), e.g., wrong numbers, low accuracy, Inf, NaN
- To check performance

Kinds of tests
- Quick incremental tests:
  - Unit tests: a few lines of code per test, check very specific properties
  - Doctests: usage examples embedded in documentation
  - Regression tests: whenever a bug is found, create a (unit) test
- Time-consuming tests:
  - Interactive tests
  - Nightly tests: automated, scheduled after office hours, time-consuming
  - Integration tests: interactions among sub-systems
  - Stress/Load/Performance tests: Big data (inputs, outputs), many hours, simulated/random data, real-world data collection
Minimal MATLAB tests

- “Mathematically equal” \((x \equiv \hat{x})\) should be “numerically equal” with no division \((\|x - \hat{x}\| \leq O(\varepsilon \|x\|))\)

- Printing results using `sprintf`, `display` are too passive, not good enough

- Tests within an algorithm with clear success condition and helpful failure message: `assert`, `error`, `warn`, `==`, `&&`, `||`, `if`, `rand`, `eps`, `intmax`

  - Example:

    ```matlab
    function write2file(varargin)
    % Example from "doc assert". Let A be a matrix in the example below
    % >> write2file('RandomA.bin',A,'double')
    min_in = 3;
    assert(nargin >= min_in, ...
    'You must call function %s with at least %d inputs', ...
    mfilename, min_in)
    ```

  - Always checking important pre-, post-, intermediate conditions
  - May slow down execution speed

- Tests outside an algorithm in a test M-file
 Assert or Error?

- `assert` is about “truth”; `error` is about a bad possible condition
- `assert` accepts one true logical condition, one error message, variable number of format strings. `assert` usually appears by itself
- `error` accepts one optional string ID, one error message, variable number of format strings. `error` has to work with `if`, or graceful exception handling constructs such as `try`, `catch`, `throw`, and `rethrow`
- `warning` is a less severe form of `error`. Example from `funappx_g.m`

```matlab
if out_param.exceedbudget == 1;
    n = 1 + (n-1)/m*floor((out_param.nmax-1)*m/(n-1));
    warning('MATLAB:funappx_g:exceedbudget', ...
        'funappx_g attempted to exceed the cost bugdet. The answer may be un
    end;
```
Unit Tests

Tests themselves have to be checked

Collection of small and quick tests outside an algorithm in M-files

MATLAB introduced since Release 2013a (Before, we used xUnit Package)

Object-oriented programming: classes with associated public/private data and methods, polymorphism, inheritance, encapsulation

Key “assertion” methods: VerifyEqual, verifyError, VerifyLessThan, verifyLessThanOrEqual

Informative test report

Properties of a good unit test:
  ▶ demonstrative
  ▶ run in less than 1 second
  ▶ always return OK
Our First Unit Test

classdef OneSolverTest < matlab.unittest.TestCase
    OneSolverTest tests solutions to the quadratic equation
    % a*x^2 + b*x + c = 0
    methods (Test)
        function testRealSolution(testCase)
            actSolution = quadraticSolver(1,-3,2);
            expSolution = [2,1];
            testCase.verifyEqual(actSolution,expSolution);
        end
    end
end
Adding a Unit Test

- For each algorithm, we need more than one unit test

Example

classdef SolverTest < matlab.unittest.TestCase
    % SolverTest tests solutions to the quadratic equation
    % a*x^2 + b*x + c = 0
    methods (Test)
        function testRealSolution(testCase)
            actSolution = quadraticSolver(1,−3,2);
            expSolution = [2,1];
            testCase.verifyEqual(actSolution,expSolution);
        end
        function testImaginarySolution(testCase)
            actSolution = quadraticSolver(1,2,10);
            expSolution = [−1+3i, −1−3i];
            testCase.verifyEqual(actSolution,expSolution);
        end
    end
end

To run: run(SolverTest)
Interpreting Unit Test Results

```plaintext
>> run(SolverTest)
Running SolverTest
..
Done SolverTest

ans =

1x2 TestResult array with properties:

Name
Passed
Failed
Incomplete
Duration

Totals:
2 Passed, 0 Failed, 0 Incomplete.
0.11967 seconds testing time.
```
Building a Unit Test Suite

- For each guaranteed algorithm in GAIL, we have one unit test file `ut_<algo>_g.m`.
- All unit test files for GAIL algorithms are saved in the subdirectory `UnitTests`.
- We have one M-file called `runtests.m` that executes all unit tests.
- An alternative to run a unit test suite in a folder:

```matlab
import matlab.unittest.TestSuite;
suiteDir = TestSuite.fromFolder(pwd);
result = run(suiteDir)
```
Doctests

- Testing code is necessary but insufficient
- Also want quick and small tests for checking correctness of examples in our documentation
- We use Thomas Smith’s open-source Doctest Package
- We package and distribute Doctest with GAIL in ThirdParty/doctest-for-matlab
- For each guaranteed algorithm in GAIL, we have one doctest for each example in its documentation
- We put our GAIL doctest M-files in UnitTests and use runtests.m to execute them
- A few things we have learnt about doctest:
  - Use `>>>` to indicate the beginning of a doctest
  - A doctest need to be followed by at least two empty comment lines
  - A long doctest can be contained in multiple lines
  - Wildcard (`***`) can be used
Doctests for the First Two Examples in meanMC_g

Examples

Example 1:
Calculate the mean of x^2 when x is uniformly distributed in [0,1], with the absolute error tolerance = 1e−2.

>> in_param.abstol=1e−2; in_param.alpha = 0.01; Yrand=@(n) rand(n,1).^2;
>> mu=meanMC_g(Yrand,in_param)
mu = 0.3***

Example 2:
Using the same function as example 1, with the absolute error tolerance 1e−2.

>> mu=meanMC_g(Yrand,1e−2)
mu = 0.3***
Running Doctests of meanMC_g

```
>> doctest meanMC_g
TAP version 13
1..4
ok 1 − "in_param.abstol=1e−2; in_param.alpha = 0.01; Yrand=@(n) rand(n,1).ˆ2;"
ok 2 − "mu=meanMC_g(Yrand,in_param) "
ok 3 − "mu=meanMC_g(Yrand,1e−2) "
ok 4 − "mu=meanMC_g(Yrand,'abstol',1e−2,'alpha',0.01) "
```
Stress Tests in GAIL

GAIL stress tests are found in the subdirectory `Workouts`, e.g., `univariate_integration_workouts/workout_integral.g.m`.

Use of random input data in `nrep` number of tests resulting in random output data:

<table>
<thead>
<tr>
<th>ninit</th>
<th>Probability In Cone</th>
<th>Success No Warning</th>
<th>Success Warning</th>
<th>Failure No Warning</th>
<th>Failure Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00% → 25.37%</td>
<td>25.18%</td>
<td>0.19%</td>
<td>74.63%</td>
<td>0.00%</td>
</tr>
<tr>
<td>100</td>
<td>23.01% → 57.56%</td>
<td>55.81%</td>
<td>1.75%</td>
<td>42.44%</td>
<td>0.00%</td>
</tr>
<tr>
<td>1000</td>
<td>56.81% → 88.45%</td>
<td>68.06%</td>
<td>20.54%</td>
<td>11.40%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Reproducibility when random data/algorithms are involved: reproduce distributions

Image credit: GAIL stress tests are found in the subdirectory `Workouts`, e.g., `univariate_integration_workouts/workout_integral.g.m`.

Use of random input data in `nrep` number of tests resulting in random output data:

<table>
<thead>
<tr>
<th>ninit</th>
<th>Probability In Cone</th>
<th>Success No Warning</th>
<th>Success Warning</th>
<th>Failure No Warning</th>
<th>Failure Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00% → 25.37%</td>
<td>25.18%</td>
<td>0.19%</td>
<td>74.63%</td>
<td>0.00%</td>
</tr>
<tr>
<td>100</td>
<td>23.01% → 57.56%</td>
<td>55.81%</td>
<td>1.75%</td>
<td>42.44%</td>
<td>0.00%</td>
</tr>
<tr>
<td>1000</td>
<td>56.81% → 88.45%</td>
<td>68.06%</td>
<td>20.54%</td>
<td>11.40%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Reproducibility when random data/algorithms are involved: reproduce distributions

Image credit: GAIL stress tests are found in the subdirectory `Workouts`, e.g., `univariate_integration_workouts/workout_integral.g.m`.

Use of random input data in `nrep` number of tests resulting in random output data:

<table>
<thead>
<tr>
<th>ninit</th>
<th>Probability In Cone</th>
<th>Success No Warning</th>
<th>Success Warning</th>
<th>Failure No Warning</th>
<th>Failure Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00% → 25.37%</td>
<td>25.18%</td>
<td>0.19%</td>
<td>74.63%</td>
<td>0.00%</td>
</tr>
<tr>
<td>100</td>
<td>23.01% → 57.56%</td>
<td>55.81%</td>
<td>1.75%</td>
<td>42.44%</td>
<td>0.00%</td>
</tr>
<tr>
<td>1000</td>
<td>56.81% → 88.45%</td>
<td>68.06%</td>
<td>20.54%</td>
<td>11.40%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
TDD Best Practice

Every time a new feature or API is introduced, unit/doc tests are added.
Every time the code is changed, unit/doc tests are executed.
Every time a bug is found, unit/doc tests are expanded.
Every time before code is checked into a repository, unit/doc tests are run.
Every time before software is released, stress tests are carried out.
Preview

Next Wed:

► HW5 due
► Fred on packaging
Reliable Mathematical Software = a Tesla

Excellent numerical software is like a well-designed vehicle that takes you from the mathematically formulated problem to the solution. With a pencil and paper we can “walk” to a solution. With a calculator we can “bike” to a solution, but for some solutions we need a car, an ATV, a plane, or a rocket:

▶ To get you to your destination
▶ To be quick and efficient
▶ To be safe and reliable
▶ To be comfortable and convenient

A well-designed vehicle is like the staunch scientific software or reliable mathematical software which this class aims to inspire.
You Have Finished Your Software Project, Now What?

You have worked hard to produce a software package that

- Is theoretically justified
- Has an optimal computational for the problem
- Is efficiently implemented
- Is thoroughly tested
- Is clearly documented
- Has a convenient user interface
- Gives instructive warnings and error messages, and
- Allows parameters to be tuned as needed

What else needs to be done?
You Have Finished Your Software Project, Now What? cont’d

You have worked hard to produce a wonderful software package. What else needs to be done?

▶ Provide illuminating workouts or case studies
▶ Package it up
▶ Make it easy to download and install
▶ Publicize and publish it
▶ Prepare for the next release
Case Studies

Besides the toy problems that you may provide in unit tests or doctests, the user would like to see your algorithm perform under realistic conditions. Thus, you should provide workouts or case studies drawn from application areas. They

▶ Show the full strength of your algorithm
▶ Compare your algorithm with competitors
▶ Show the user how to set required and optional inputs

For GAIL we are working on Monte Carlo financial examples, mostly option pricing, using `meanMC_g.m`. These both serve to show new user what can be done, but also are helpful in teaching MATH 565 Monte Carlo Methods in Finance.

For `funappx_g.m` and `integral_g.m` and we need more realistic examples.
Packaging

Your new software package needs to be put in a rational directory structure for distribution.

▶ This may be similar to the development directory structure, but there might be some differences.
▶ Each folder needs a contents file.
▶ This needs to work no matter where the directory sits on the user’s computer.

Do not wait to release until your software is perfect, or it will never be released. Release it when it does something good that you want to share with others.
Your new software package needs to be easy to get.

- Zip the whole directory into one file.
- Put the zipped file on a host where it can be downloaded, e.g., GAIL code site or math.iit.edu ftp site.
- Create a (MATLAB) script to download and run the installation program, e.g., `DownloadInstallGail_1_0_0.m`
%This mfile may be used to download and install GAIL into
% the location you choose
%
% Step 1. Place this .m file where you want GAIL to go
%
% Step 2. Run this mfile

%% Download the package and change the directory
disp('The GAIL package is now downloading ...')
unzip('https://gail.googlecode.com/files/GAIL_1.0.0.zip') %download and unzip
cd('GAIL_1.0.0') %get to the right subdirectory
cd('GAIL_Matlab') %get to the right subdirectory

%% Install Gail
GAIL_Install %this installs GAIL
fprintf('

Next we will run a quick test. Press return to continue...
')
pause
DownloadInstallGail_1_0_0.m Example

%%% Run a quick test
fprintf('\n\nmuhat=meanMC_g(@(n) rand(n,1))\n')
muhat=meanMC_g(@(n) rand(n,1)) %run meanMC_g once
fprintf('\nThis answer should be close to 0.5.\n\n')
% Then you should be ready to use GAIL
disp('Next README.txt will be displayed. Press return to continue...')
pause

%%% Printing out README.txt
more on
type('README.txt')
more off
Installing

After the package is downloaded onto the user’s computer at the right location, the package may need to be installed. The installer script, e.g., GAIL_Install.m should:

- Check for compatibility of the package with the user environment
- Set paths
- Move files, if necessary
- Tell the user whether the software was successfully installed
GAIL_Install.m Example

% GAIL_Install.m
% This script is to install GAIL, to add GAIL path to MATLAB search path.
clear all; close all; clc;
[GAILPATH,GAILVERSION,PATHNAMESEPARATOR,MATLABVERSION] = GAILstart;
fprintf('
Welcome to GAIL version %g.
', GAILVERSION);
if MATLABVERSION < 7,
    error('This version is only supported on Matlab 7.x and above.');
else
    gailp=genpath(GAILPATH); % adding all subdirectories
end
addpath(gailp); % Add GAIL directories and subdirectories
savepath; % Save the changes
fprintf('
GAIL version %g has been installed successfully.
', GAILVERSION);
Publicizing and Publishing

People need to know about your package.

- Announce it on mailing lists such as Numerical Analysis (NA) Digest, which is distributed weekly
- Speak about it at conferences such as Midwest Numerical Analysis Day, SIAM Annual Meeting, Joint Mathematics Meetings, biennial Monte Carlo and Quasi-Monte Carlo conferences
- Publish it in software oriented journals such as ACM Transactions on Mathematical Software
- Encourage those who use it to cite the software as a publication. Give a suggested citation in your documentation and on your project site. Cite it in your own publications.
Prepare for the Next Release

As you and others use your package, you will find ways to make it more wonderful that it already is.

- On the project page maintain a list of suggestions for corrections and improvements.
- Find partners to share the work in maintaining and improving the software.
- Assign items to those who will share future development.

Some of the most popular software has undergone multiple releases: MATLAB ??????????, Mathematica ??????, NAG Library ?????????, JMP ??????, R ????
Prepare for the Next Release

As you and others use your package, you will find ways to make it more wonderful that it already is.

▶ On the project page maintain a list of suggestions for corrections and improvements.
▶ Find partners to share the work in maintaining and improving the software.
▶ Assign items to those who will share future development.

Some of the most popular software has undergone multiple releases: MATLAB 8.2 (2013b), Mathematica 9.0.1, NAG Library Mark 23, JMP 10.0.2, R 3.0.2
References I


References II


References III


